Robust Plane Sweep for Intersecting Segments

Jean-Daniel Boissonnat, Franco P. Preparata

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Robust plane sweep for intersecting segments

Jean-Daniel Boissonnat — Franco P. Preparata

N° 3270

Septembre 1997

THÈME 2

Rapport de recherche

ISSN 0249-6399

Robust plane sweep for intersecting segments

Jean-Daniel Boissonnat* , Franco P. Preparata†

Thème 2 — Génie logiciel
et calcul symbolique

Projet Prisme

Rapport de recherche n° 3270 — Septembre 1997 — 35 pages

Abstract: In this paper, we reexamine in the framework of robust computation the Bentley-Ottmann algorithm for reporting intersecting pairs of segments in the plane. This algorithm has been reported as being very sensitive to numerical errors. Indeed, a simple analysis reveals that it involves predicates of degree 5, presumably never evaluated exactly in most implementation. Within the exact-computation paradigm we introduce two models of computation aimed at replacing the conventional model of real-number arithmetic. The first model (predicate arithmetic) assumes the exact evaluation of the signs of algebraic expressions of some degree, and the second model (exact arithmetic) assumes the exact computation of the value of

This work was partially supported by ESPRIT LTR 21957 (CGAL) and by the U.S. Army Research Office under grant DAAH04-96-1-0013. Part of the work was done while J-D. Boissonnat was visiting the Center for Geometric Computing at Brown University.

* E-mail: Jean-Daniel.Boissonnat@sophia.inria.fr

† Center for Geometric Computing, Computer Science Department, Brown University, 115 Waterman Street, Providence, RI 02912-1910, USA. E-mail: franco@cs.brown.edu

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : 04 93 65 77 77 - International : +33 4 93 65 77 77 — Fax : 04 93 65 77 65 - International : +33 4 93 65 77 65
à partir du 01/01/1998
Téléphone : 04 92 38 77 77 - International : +33 4 92 38 77 77 — Fax : 04 92 38 77 65 - International : +33 4 92 38 77 65

such (bounded-degree) expressions. We identify the characteristic geometric property enabling the correct report of all intersections by plane sweeps. Verification of this property involves only predicates of (optimal) degree 2, but its straighforward implementation appears highly inefficient. We then present algorithmic variants that have low degree under these models and achieve the same performance as the original Bentley-Ottmann algorithm. The technique is applicable to a more general case of curved segments.

Key-words: Computational Geometry, Robustness, Sweep line algorithms, Exact arithmetic, Bentley-Ottmann's algorithm

(Résumé : tsvp)

Algorithmes robustes par balayage pour rechercher des segments qui se coupent

Résumé : Dans cet article, nous réexaminons, du point de vue de la robustesse, l'algorithme de Bentley-Ottmann qui recherche dans une famille finie de segments du plan les paires qui se coupent. Cet algorithme est bien connu pour être très sensible aux erreurs numériques, ce que confirme une analyse qui montre qu'il demande d'évaluer des prédicats de degré 5. Dans le cadre du calcul exact, nous introduisons deux modèles de calcul qui visent à remplacer le modèle habituel de l'arithmétique réelle. Le premier modèle (arithmétique des prédicats) suppose que l'on peut évaluer de manière exacte le signe de toute expression algébrique de degré au plus d. Le deuxième modèle (arithmétique exacte) suppose que la valeur de telles expressions peut être calculée exactement. On met en évidence la propriété caractéristique qui garantit le bon comportement d'un algorithme de balayage. Vérifier que cette propriété est satisfaite ne requiert que des prédicats de degré 2 mais ne conduit pas directement à un algorithme efficace. On présente trois variantes de l'algorithme de Bentley-Ottmann de même complexité que celui-ci mais dont les degrés sont plus faibles. La technique s'applique à certains types d'arcs de courbe.

Mots-clé : Géométrie algorithmique, Robustesse, Balayage, Arithmétique exacte, Algorithme de Bentley-Ottmann

1 Introduction

As is well known, Computational Geometry has traditionally adopted the arithmetic model of exact computation over the real numbers. This model has been extremely productive in terms of algorithmic research, since it has permitted a vast community to focus on the elucidation of the combinatorial (topological) properties of geometric problems, thereby leading to sophisticated and efficient algorithms. Such approach, however, has a substantial shortcoming, since all computer calculations have finite precision, a feature which affects not only the quality of the results but even the validity of specific algorithms. In other words, in this model algorithm correctness does not automatically translate into program correctness. In fact, there are several reports of failures of implementations of theoretically correct algorithms (see e.g. [For87, Hof89]). This state-of-affairs has engendered a vigorous debate within the research community, as is amply documented in the literature. Several proposals have been made to remedy this unsatisfactory situation. They can be split into two broad categories according to whether they perform exact computations (see, e.g., [BKM+95, FV93, Yap97, She96]) or approximate computations (see, e.g., [Mil88, HHK89, Mil89]).

This paper fine-tunes the exact-computation paradigm. The numerical computations of a geometric algorithm are basically of two types: tests (predicates) and constructions, with clearly distinct roles. Tests are associated with branching decisions in the algorithm that determine the flow of control, whereas constructions are needed to produce the output data. While approximations in the execution of constructions are often acceptable, approximations in the execution of tests may produce incorrect branching, leading to the inconsistencies which are the object of the criticisms leveled against geometric algorithms. The exact-computation paradigm therefore requires that tests be executed with total accuracy. This will guarantee that the result of a geometric algorithm will be topologically correct albeit geometrically approximate. This also means that robustness is in principle achievable if one is willing to employ the required precision. The reported failures of structurally correct algorithms are entirely attributable to non-compliance with this criterion.

Therefore, geometric algorithms can also be characterized on the basis of the complexity of their predicates. The complexity of a predicate is expressed by the degree of a homogeneous polynomial embodying its evaluation. The degree of an algorithm is the maximum degree of its predicates, and an algorithm is robust if the adopted precision matches the degree requirements.

The "degree criterion" is a design principle aimed at developing low-degree algorithms. This approach involves re-examining under the degree criterion the rich body of geometric algorithms known today, possibly without negatively affecting traditional algorithmic efficiency. A previous paper [LPT96] considered as an illustration of this approach the issue of proximity queries in two and three dimensions. As an additional case of degree-driven algorithm design, in this paper we confront another class of important geometric problems, which have caused considerable difficulties in actual implementations: plane-sweep problems for sets of segments. As we shall see, plane-sweep applications involve a number of predicates of different degree and algorithmic power. Their analysis will lead not only to new and robust implementations (an outcome of substantial practical interest) but elucidate on a theoretical level some deeper issues pertaining to the structure of several related problems and the mechanism of plane-sweeps.

2 Plane sweep of intersecting segments

Given is a finite set $\mathcal{S}$ of line segments in the plane. Each segment is defined by the coordinates of its two endpoints. We discuss the three following problems (see Figure 1) :

Pb1 : report the pairs of segments of $\mathcal{S}$ that intersect .

Pb2 : construct the arrangement $\mathcal{A}$ of $\mathcal{S}$, i.e., the incidence structure of the graph obtained interpreting the union of the segments as a planar graph.

Pb3: construct the trapezoidal map $\mathcal{T}$ of $\mathcal{S}$. $\mathcal{T}$ is obtained by drawing two vertical line segments (*walls*), one above and one below each endpoint of the segments and each intersection point. The walls are extended either until they meet another segment of $\mathcal{S}$ or to infinity.

Let $S_1, \ldots, S_n$ be the segments of $\mathcal{S}$ and let k be the number of intersecting pairs. We say that the segments are in *general position* if any two intersecting segments intersect in a single point, and all endpoints and intersection points are distinct.

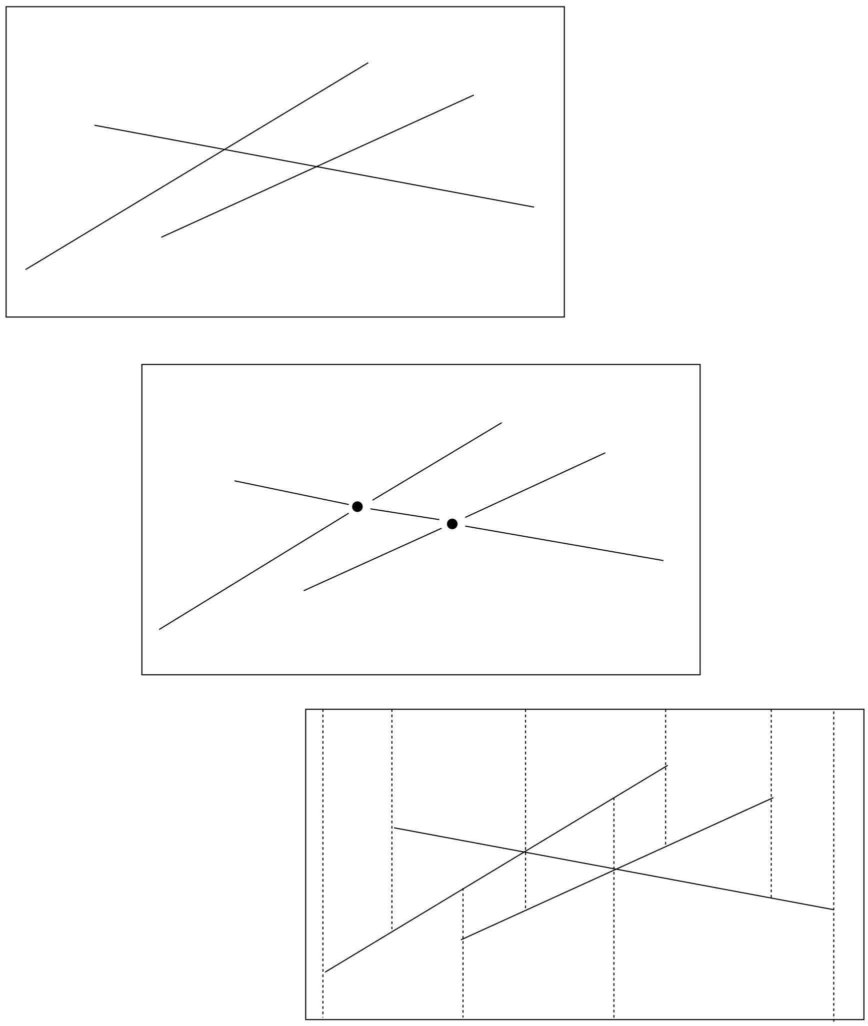

Figure 1: $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{T}$.

The number of intersection points is no more than the number of intersecting pairs of segments and both are equal if the segments are in general position. Therefore, the number of vertices of $\mathcal{A}$ is at most k, the number of edges of $\mathcal{A}$ is at most $n+2k$ and the number of vertical walls in $\mathcal{T}$ is at most $2(n+k)$, the bounds being tight when the segments are in general position. Thus the sizes of both $\mathcal{A}$ and $\mathcal{T}$ are $O(n+k)$. We didn't consider here the 2-dimensional faces of either $\mathcal{A}$ or $\mathcal{T}$. Including them would not change the problems we address.

3 Algebraic degree and arithmetic models

It is well known that the efficient algorithms that solve Pb1-Pb3 are very unstable when implemented as programs, and several frustrating experiences have been reported [For85]. This motivates us to carefully analyze the predicates involved in those algorithms. We first introduce here some terminology borrowed from [LPT96]. We consider each input data (i.e., coordinates of an endpoint of some segment of $\mathcal{S}$) as a *variable*.

An *elementary predicate* is the sign $-$, 0, or $+$ of a homogeneous multivariate polynomial whose arguments are a subset of the input variables. The degree of an elementary predicate is defined as the maximum degree of the irreducible factors (over the rationals) of the polynomials that occur in the predicate and that do not have a constant sign. A *predicate* is more generally a boolean function of elementary predicates. Its degree is the maximum degree of its elementary predicates.

The *degree of an algorithm* A is defined as the maximum degree of its predicates.

The *degree of a problem* P is defined as the minimum degree of any algorithm that solves P.

In most problems in Computational Geometry, $d = O(1)$. However, as d affects the speed and/or robustness of an algorithm, it is important to measure d precisely.

In the rest of this paper we consider the degree as an additional measure of algorithmic complexity. Note that qualitatively degree and memory requirement are similar, since the arithmetic capabilities demanded by a given degree must be available, albeit they may be never resorted to in an actual run of the algorithm (since the input may be such that predicates may be evaluated reliably with lower precision).

We will consider two arithmetic models. In the first one, called the *predicate arithmetic of degree d*, the only numerical operations that are allowed are the evaluations of predicates of degree at most d. Algorithms of degree d can therefore be implemented exactly in the predicate arithmetic model of degree d. This model is motivated by recent results that show that evaluating the sign of a polynomial expression may be faster than computing its value (see [ABD$^+$97, BY97, BEPP97, Cla92, She96]). This model is however very conservative since the non-availability of the arithmetics required by a predicate is assimilated to an entirely random choice of the value of the predicate.

The second model, called the *exact arithmetic of degree d*, is more demanding. It assumes that values (and not just signs) of polynomials of degree at most d be represented and computed exactly (i.e., roughly as d-fold precision integers). However, higher-degree operations (e.g., a multiplication one of whose factors is a d-fold precision integer) are appropriately rounded. Typical rounding is rounding to the nearest representable number but less accurate rounding can also be adequate as will be demonstrated later. Let A be an algorithm of degree d. If each input data is a b-bit integer, the size of each monomial occuring in a predicate of A is upper bounded by $2^{(b+1)d}$. Moreover, let v be the number of variables that occur in a predicate; for most geometric problems and, in particular, for those considered in this paper, v is a small constant. It follows that an algorithm of degree d requires precision $p \leq d(b+1+\log v)$ in the exact arithmetic model of degree d.

4 The predicates for Pb1-Pb3

We use the following notations. The coordinates of point A_i are denoted x_i and y_i. $A_i <_x A_j$ means that the x-coordinate of point A_i is smaller than the x-coordinate of point A_j. Similarly for $<_y$. $[A_iA_j]$ denotes the line segment whose left and right endpoints are respectively A_i and A_j, while (A_iA_j) denotes the line containing $[A_iA_j]$. $A_i <_y (A_jA_k)$ means that point A_i lies below line (A_jA_k).

4.1 Predicates

Pb1 only requires that we check if two line segments intersect (Predicate $2'$ below).

Pb2 requires in addition the ability to sort intersection points along a line segment (Predicate 4 below).

Pb3 requires the ability to execute all the predicates listed below :

Predicate 1 : $A_0 <_x A_1$

Predicate 2 : $A_0 <_y (A_1A_2)$

Predicate 2′ : $[A_0A_1]\bigcap[A_2A_3] \neq \emptyset$

Predicate 3 : $A_0 <_x [A_1A_2]\bigcap[A_3A_4]$

Predicate 4 : $[A_0A_1]\bigcap[A_2A_3] \ <_x \ [A_0A_1]\bigcap[A_4A_5]$

Predicate 5 : $[A_0A_1]\bigcap[A_2A_3] <_x [A_4A_5]\bigcap[A_6A_7]$

Two other predicates appear in some algorithms that report segment intersections :

Predicate 3′ : $(x = x_0)\bigcap[A_1A_2] <_y (x = x_0)\bigcap[A_3A_4]$

Predicate 4′ : $[A_0A_1]\bigcap[A_2A_3] <_y (A_4A_5)$

4.2 Algebraic degree of the predicates

We now analyze the algebraic degree of the predicates introduced above.

Proposition 1 *The degree of Predicates i and i' ($i = 1, \ldots, 5$) is i.*

Proof. We first provide explicit formulae for the predicates.

Predicate 2 (orientation test) : $A_0 <_y (A_1A_2)$

Evaluating Predicate 2 is equivalent to evaluating the sign of :

$$\mathrm{orient}(A_0A_1A_2) = \begin{vmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{vmatrix}$$

Predicate 2′ (intersection test) : $[A_0A_1]\bigcap[A_2A_3] \neq \emptyset$

Predicate 2′ can be implemented as follows for the case $A_0 <_x A_2$ (otherwise we exchange the roles of $[A_0A_1]$ and $[A_2A_3]$) :

if $A_3 <_x A_1$
    if orient$(A_0A_1A_2) \times$ orient$(A_0A_1A_3) < 0$ **then return true**
    else return false
else
    if orient$(A_0A_1A_2) \times$ orient$(A_2A_3A_1) > 0$ **then return true**
    else return false

Therefore, in all cases, Predicate $2'$ reduces to Predicate 2.

The intersection point $I = [A_iA_j] \bigcap [A_kA_l]$ is given by :

$$I = A_i + (A_j - A_i)\frac{N_I}{D_I} \tag{1}$$

with $N_I = \text{orient}(A_iA_kA_l)$ and

$$\begin{aligned} D_I &= \begin{vmatrix} x_j - x_i & x_k - x_l \\ y_j - y_i & y_k - y_l \end{vmatrix} \\ &= \text{orient}(A_iA_jA_k) - \text{orient}(A_iA_jA_l) \\ &\stackrel{def}{=} \text{Orient}(A_iA_jA_kA_l) \end{aligned}$$

Predicate 4′ : $[A_2A_3] \bigcap [A_4A_5] <_y (A_0A_1)$

Predicate 4 reduces to evaluating orient(I, A_0, A_1) where I is the intersection of $[A_2A_3]$ and $[A_4A_5]$. It follows from (1) that this is equivalent to evaluating the sign of orient$(A_2A_3A_4A_5)$ and of

$$\text{orient}(A_0A_1A_2) \times \text{Orient}(A_2A_3A_4A_5) - \text{orient}(A_2A_4A_5) \times \text{Orient}(A_0A_1A_2A_3)$$

Predicates 3-5 :

Explicit formulas for Predicates 3, 4 and 5 can be immediately deduced from the coordinates of the intersection points $I = [A_0A_1] \bigcap [A_2A_3]$ and $J = [A_4A_5] \bigcap [A_6A_7]$ which are given by (1). If $A_4A_5 = A_0A_1$, it is clear from Equation (1) that $(x_1 - x_0)$ is a common factor of $x_I - x_0$ and $x_J - x_0$.

Predicate 3′ : $(x = x_0) \bigcap [A_1A_2] <_y (x = x_0) \bigcap [A_3A_4]$

If $[A_1A_2]$ and $[A_3A_4]$ do not intersect, Predicate $3'$ reduces to Predicate 2. Otherwise, it reduces to Predicate 3.

The above discussion shows that the degree of predicates i and i' is *at most* i. To establish that it is *exactly* i, we have shown in Appendix that the polynomials of Predicates 2, 3, $4'$ and 5, as well as the factor other than $(x_1 - x_0)$ involved in Predicate 4, are irreducible over the rationals.

It follows that the proposition is proved for all predicates.

□

Recalling the requirements of the various problems in terms of predicates, we have :

Proposition 2 *The algebraic degrees of Pb1, Pb2 and Pb3 are respectively 2, 4 and 5.*

4.3 Implementation of Predicate 3 with exact arithmetic of degree 2

As it will be useful in the sequel, we show how to implement Predicate 3 (of degree 3) under the exact arithmetic of degree 2. From Equation (1) we know that Predicate 3 can be written as :

$$(x_0 - x_1)\ \mathrm{Orient}(A_1A_2A_3A_4) < (x_2 - x_1)\ \mathrm{orient}(A_1A_3A_4). \tag{2}$$

For convenience, let $A = \mathrm{orient}(A_1A_3A_4)$, $B = \mathrm{Orient}(A_1A_2A_3A_4)$, $x_{01} = x_0 - x_1$ and $x_{21} = x_2 - x_1$.

We stipulate to employ floating point arithmetic conforming to the IEEE 754 standard [Gol91]. In this standard, simple precision allows us to represent b-bit integers with $b = 24$ and double precision allows us to represent b'-bit integers with $b' = 2b + 5 = 53$. The coordinates of the endpoints of the segments are represented in simple precision and the computations are carried out in double precision. We denote $\oplus$, $\otimes$ and $\oslash$ the rounded arithmetic operations $+, \times$ and $/$. In the IEEE 754 standard, all four arithmetic operations are exactly rounded, i.e., the computed result is the floating point number that best approximates the exact result.

Since Orient$(A_1A_2A_3A_4)$ and orient$(A_1A_3A_4)$ are polynomials of degree 2, the four terms x_{01}, x_{21}, A and B in Inequality (2) can be computed exactly and the following monotonicity property is a direct consequence of exact rounding of arithmetic operations :

Monotonicity property 1 : $x_{01} \otimes B < x_{21} \otimes A \Longrightarrow x_{01} \times B < x_{21} \times A$.

This implies that the comparison between the two computed expressions $x_{01} \otimes B$ and $x_{21} \otimes A$ evaluates Predicate 3 except when these numbers are equal.

In most algorithms, an intersection point is compared with many endpoints. It is therefore more efficient to compute and store the coordinates of each intersection point and to perform comparisons with the computed abscissae rather than evaluating (2) repeatedly. We now illustrate an effective rounding procedure of the x-coordinates of intersection points.

Lemma 3 *If the coordinates of the endpoints of the segments are simple precision integers, then the abscissa x_I of an intersection point can be rounded to one of its two nearest simple precision integers using only double precision floating point arithmetic operations.*

Proof. We assume that the coordinates of the endpoints of the segments are represented as b-bit integers stored as simple precision floating point numbers. The computations are carried out in double precision.

The rounded value $\tilde{x}_I$ of x_I is given by :

$$\tilde{x}_I = \lfloor((x_{21} \otimes A) \oslash B)\rceil \oplus x_1,$$

where $\lfloor X\rceil$ denotes the integer nearest to X (with any tie breaking rule). If $\varepsilon = 2^{-b'}$ is a strict bound to the modulus of the relative error of all arithmetic operations, $\tilde{X} = (x_{21} \otimes A) \oslash B$ satisfies the following relations :

$$\frac{x_{21}A}{B}(1-2\varepsilon) \approx \frac{x_{21}A}{B}(1-\varepsilon)^2 < \tilde{X} < \frac{x_{21}A}{B}(1+\varepsilon)^2 \approx \frac{x_{21}A}{B}(1+2\varepsilon).$$

As $\frac{x_{21}A}{B} = x_I - x_1 \leq 2^{b+1}$, we obtain

$$|(x_{21} \otimes A) \oslash B - x_{21}A/B)| \lesssim 2^{b+1}22^{-b'} = 2^{-b-3} \ll 1.$$

We round $\tilde{X}$ to the nearest integer $\lfloor \tilde{X} \rceil$. Since $\lfloor \tilde{X} \rceil$ and x_1 are $(b+1)$-bit integers, there is no error in the addition. Therefore, $\tilde{x}_I$ is a $(b+2)$-bit integer and the absolute error on $\tilde{x}_I$ is smaller than 1. □

It follows that, under the hypothesis of the lemma, if E is an endpoint, I is an intersection point and $\tilde{I}$ the corresponding rounded point, the following monotonicity property holds :

Monotonicity property 2 :

$$\tilde{I} <_x E \Longrightarrow I <_x E$$
$$E <_x \tilde{I} \Longrightarrow E <_x I$$

Notice that the monotonicity property does not necessarily hold for two intersection points.

Remark 1. A result similar to Lemma 3 has been obtained by Priest [Pri92] for points with floating point coordinates. More precisely, if the endpoints of the segments are represented as simple precision floating-point numbers, Priest [Pri92] has proposed a rather complicated algorithm that uses double precision floating point arithmetic and rounds x_I to the nearest simple precision floating point number. This stronger result also implies the monotonicity property.

4.4 Algebraic degree of the algorithms

The naive algorithm for detecting segment intersections (Pb1) evaluates $\Theta(n^2)$ Predicates $2'$ and thus is of degree 2, which is degree-optimal by the proposition above. Although the time-complexity of the naive algorithm is worst-case optimal, since $0 \leq k \leq \frac{1}{2}n(n-1)$, it is worth looking for an output sensitive algorithm whose complexity depends on both n and k. Chazelle and Edelsbrunner [CE92] have shown that $\Omega(n \log n + k)$ is a lower bound for Pb1, and therefore also for Pb2 and Pb3. A very recent algorithm of Balaban [Bal95] solves Pb1 optimally in $O(n \log n + k)$ time using $O(n)$ space. This algorithm does not solve Pb2 nor Pb3 and, since it uses Predicate $3'$, its degree is 3.

Pb2 can be solved by first solving Pb1 and subsequently sorting the reported intersection points along each segment. This can easily be done in $O((n+k)\log n)$ time by a simple algorithm of degree 4 using $O(n)$ space. A direct (and asymptotically more efficient) solution to Pb2 has been proposed by Chazelle and Edelsbrunner [CE92]. Its time complexity is $O(n\log n + k)$ and it uses $O(n+k)$ space. Their algorithm, which constructs the arrangement of the segments, is of degree 4.

A solution to Pb3 can be deduced from a solution to Pb2 in $O(n+k)$ time using a very complicated algorithm of Chazelle [Cha91]. A deterministic and simple algorithm due to Bentley and Ottmann [BO79] solves Pb3 in $O((n+k)\log n)$ time, which is slightly suboptimal, using $O(n)$ space. This classical algorithm uses the sweep-line paradigm and evaluates $O((n+k)\log n)$ predicates of all types discussed above, and therefore has degree 5. Incremental randomized algorithms [CS89, BDS$^+$92] construct the trapezoidal map of the segments and thus solve Pb3 and have degree 5. Their time complexity and space requirements are optimal (though only as expected performances).

In this paper, we revisit the Bentley-Ottmann algorithm and show that a variant of degree 3 (instead of 5) can solve Pb 1 with no sacrifice of performance (Section 6.1). Although this algorithm is slightly suboptimal with respect to time complexity, it is much simpler than Balaban's algorithm. We also present two variants of the sweep line algorithms. The first one (Section 6.2) uses only predicates of degree at most 2 and applies to the restricted but important special case where the segments belong to two subsets of non intersecting segments. The second one (Section 7) uses the exact arithmetic of degree 2. All these results are based on a (non-efficient) lazy sweep-line algorithm (to be presented in Section 5) that solves Pb1 by evaluating predicates of degree at most 2.

Remark 2. When the segments are not in general position, the number s of intersection points can be less than the number k of intersecting pairs. In the extreme, $s = 1$ and $k = n(n-1)/2$. Some algorithms can be adapted so that their time complexities depend on s rather than k [BMS94]. However, a lower bound on the degree of such algorithms is 4 since they must be able to detect if two intersection points are identical, therefore to evaluate Predicate $4'$.

5 A lazy sweep-line algorithm

Let $\mathcal{S}$ be a set of n segments whose endpoints are $E_1, \ldots, E_{2n}$. For a succint review, the standard algorithm first sorts $E_1, \ldots, E_{2n}$ by increasing x-coordinates and stores the sorted points in a priority queue X. Next, the algorithm begins sweeping the plane with a vertical line L and maintains a data structure Y that represents a subset of the segments of $\mathcal{S}$ (those currently intersected by L, ordered according to the ordinates of their intersections with L). Intersections are detected in correspondence of adjacencies created in Y, either by insertion/deletion of segment endpoints, or by order exchanges at intersections. An intersection, upon detection, is inserted into X according to its abscissa. Of course, a given intersection may be detected several times. Multiple detections can be resolved by performing a preliminary membership test for an intersection in X and omitting insertion if the intersection has been previously recorded. We stipulate to use another policy to resolve multiple detections, namely to remove from X an intersection point I whose associated segments are no longer adjacent in Y. Event I will be reinserted in X when the segments become again adjacent in Y. This policy has also the advantage of reducing the storage requirement of Bentley-Ottmann's algorithm to $O(n)$ [Bro81].

We now describe a modification of the sweep-line algorithm that does not need to process the intersection points by increasing x-coordinates. First, the algorithm sorts the endpoints of the segments by increasing x-coordinates into an array X. Let $E_1, \ldots, E_{2n}$ be the sorted list of endpoints. The algorithm uses also a dictionary Y that stores an ordered subset of the line segments.

The algorithm rests on the definitions of *active* and *prime* pairs to be given below. We need the following notations. We denote by $L(E_i)$ the vertical line passing through E_i. Slab (E_i, E_{i+1}) denotes the open vertical slab bounded by $L(E_i)$ and $L(E_{i+1})$ and $(E_i, E_{i+1}]$ denotes the semiclosed slab obtained by adjoining line $L(E_{i+1})$ to the open slab (E_i, E_{i+1}). For two segments S_k and S_l, we denote by A_{kl} their rightmost left endpoint, by B_{kl} their leftmost right endpoint and by I_{kl} their common point when they intersect (see Figure 2). In addition, W_{kl} denotes the set of segment endpoints that belong to the (closed) region bounded by the vertical lines $L(A_{kl})$ and $L(B_{kl})$ and by the two segments (a double wedge). We denote by E_{kl} the most recently processed element of W_{kl} and by F_{kl} the element of W_{kl} to be processed next. (Note that E_{kl} and F_{kl} are always defined, since they may res-

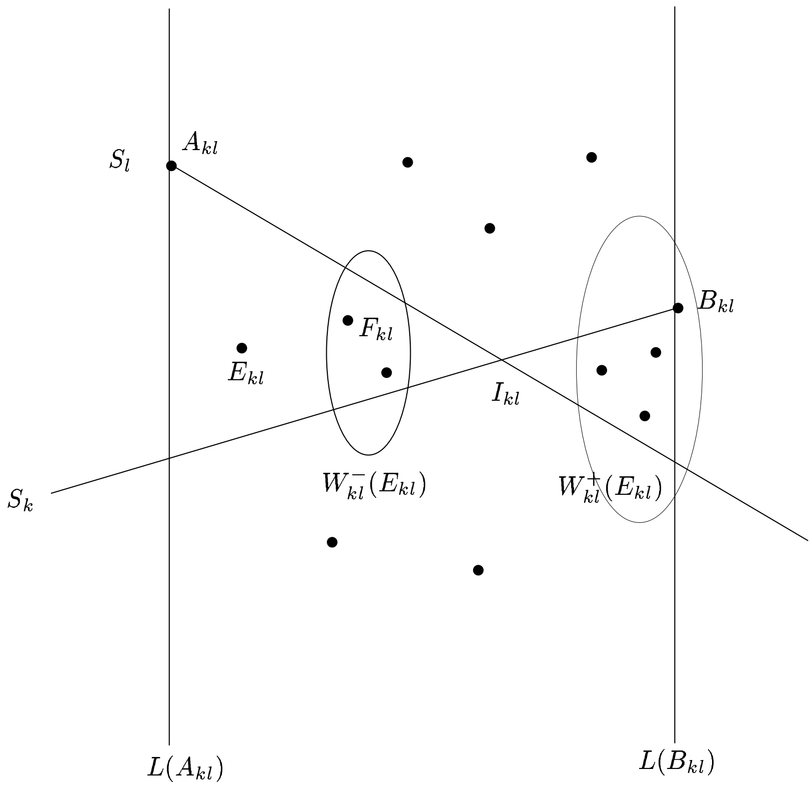

Figure 2: For the definitions of $W_{kl}^-(E_{kl})$ and $W_{kl}^+(E_{kl})$.

pectively coincide with A_{kl} and B_{kl}.) Lastly, we define sets $W_{kl}^+(E_{kl})$ and $W_{kl}^-(E_{kl})$ as follows. If S_k and S_l do not intersect, $W_{kl}^+(E_{kl}) = \emptyset$ and $W_{kl}^-(E_{kl})$ consists of all points $E \in W_{kl}$, $E_{kl} \leq_x E$. Otherwise, an endpoint $E \in W_{kl}$ belongs to $W_{kl}^+(E_{kl})$ (resp., to $W_{kl}^-(E_{kl})$) if $E_{kl} \leq_x E$ and if the slab $(E_{kl}, E]$ does (resp., does not) contain I_{kl}.

Definition 4 *Let (S_k, S_l) be a pair of segments and assume without loss of generality that $S_k \bigcap L(E_{kl}) <_y S_l \bigcap L(E_{kl})$. The pair is said to be* active *if the following conditions are satisfied :*

1. *S_k and S_l are adjacent in Y,*
2. *$S_k < S_l$ in Y,*
3. *$F_{kl} \in W_{kl}^{+}(E_{kl})$* (Emptiness condition).

Observe that the emptiness condition implies that the segments intersect.

Definition 5 *A pair of active segments (S_k, S_l) is said to be* prime *if the next element to be processed belongs to W_{kl} (therefore is $F_{kl} \in W_{kl}^{+}(E_{kl})$).*

We say that an intersection (or an active pair) is *processed* when the algorithm reports it, exchanges its members in Y, and updates the set of active pairs of segments accordingly. After sorting the segment endpoints, the lazy sweep-line algorithm works as follows. While there are active pairs, the algorithm selects any of them and processes it. When there are no more active pairs the algorithm proceeds to the next endpoint, i.e., it inserts or removes the corresponding segment in Y (as appropriate) and updates the set of active pairs. Actually, the next endpoint may be accessed once there are no more prime pairs (a subset of the active pairs), without placing any deadline on the processing of the current active pairs as long as they are not prime. When there are no more active pairs and no more endpoints to be processed, the algorithm stops.

For reasons that will be clear below, the algorithm will not be specified in the finest detail, since several different implementations are possible. The main issue is the efficient detection of active and prime pairs and several solutions, all consistent with the described lazy algorithm, will be discussed in Sections 6 and 7.

It should be noted that deciding if a pair of intersecting segments is active or prime reduces to the evaluation of Predicates 2 only. Therefore, the algorithm just described involves only Predicates 1 and 2 and is of degree 2 by Proposition 1. It should also be pointed out that two intersection points or even an intersection point and an endpoint won't necessarily be processed in the order of their x-coordinate. As a consequence, Y won't necessarily represent the ordered set of segments intersecting some vertical line L (as in the standard algorithm).

Let $Y^{-}(E_i)$ and $Y^{+}(E_i)$ be, respectively, snapshots of the data structure Y immediately before and after processing event E_i, $i = 1, \ldots, 2n$. Observe that $Y^{-}(E_i)$

and $Y^+(E_i)$ differ only by the segment S that has E_i as one of its endpoints. Let $Y(E_i) = Y^-(E_i) \bigcup Y^+(E_i)$. The order relation in Y is denoted by $<$.

Theorem 6 *If Predicates 1 and 2 are evaluated exactly, the described lazy sweep-line algorithm will detect all pairs of segments that intersect.*

Proof : The algorithm (correctly) sorts the endpoints $E_1, \ldots, E_{2n}$ of the segments by increasing x-coordinates into X. Consequently, the set of segments that intersect $L(E)$ and the set of segments in $Y(E)$ coincide for any endpoint E. The proof of the theorem is articulated now as two lemmas and their implications.

Lemma 7 *Two segments have exchanged their positions in Y if and only if they intersect and if the pair has been processed.*

Proof. Let us consider two segments, say S_k and S_l, that do not intersect. Without loss of generality, let $S_k < S_l$ in $Y(A_{kl})$. Assume for a contradiction that $S_l < S_k$ in $Y^-(B_{kl})$. S_k and S_l cannot exchange their positions because they will never form an active pair, Therefore, $S_l < S_k$ in $Y^-(B_{kl})$ can only happen if there exists a segment S_m, $m \neq k, l$, that at some stage in the execution of the algorithm was present in Y together with S_k and S_l and caused one of the following two events to occur :

1. $S_m > S_l$ and the positions of S_m and S_k are exchanged in Y
2. $S_m < S_k$ and the positions of S_m and S_l are exchanged in Y.

In both cases, the segments that exchange their positions are not consecutive in Y, violating Condition 1 of Definition 4.

Therefore, two segments can exchange their positions in Y only if they intersect and this can only happen when their intersection is processed. Moreover, when the intersection has been processed, the segments are no longer active and cannot exchange their positions a second time. $\square$

We say that an endpoint E of S is *correctly placed* if and only if the subset of the segments that are below E (in the plane) coincides with the subset of the segments $< S$ in $Y(E)$, i.e.,

$$\forall S' \in Y(E), \quad S' < S \Longleftrightarrow S' \bigcap L(E) <_y E = S \bigcap L(E).$$

Otherwise, E is said to be *misplaced*.

Lemma 8 *If Predicates 1 and 2 are evaluated exactly, both endpoints of every segment are correctly placed.*

Proof. Assume, for a contradiction, that E of S is the *first* endpoint to be misplaced by the algorithm.

Claim 1 *E can be misplaced only if there exist at least two intersecting segments S_k and S_l in $Y^-(E)$ such that E belongs to W_{kl}.*

Proof. First recall that Predicate 2 is the only predicate involved in placing S in Y.

Consider first the case where E is the left endpoint of S. For any pair (S_k, S_l) of segments in $Y^-(E)$, for which S is either above or below both S_k and S_l, the relative position of S with respect to S_k and S_l does not depend on the relative order of S_k and S_l in $Y^-(E)$. Therefore, E will be correctly placed in $Y^+(E)$.

If E is a right endpoint, since the left endpoint of S has been correctly placed, E can only be misplaced if there exists a segment $S' \in Y^-(E)$ intersecting S such that the relative positions of S and S' in $Y^+(A)$ and $Y^-(E)$ are the same (A is the rightmost left endpoint of S and S') while this change has not been executed by the algorithm in Y, i.e. S and S' have the same relative position in $Y^+(A)$ and $Y^-(E)$. In that case, $E = B_{kl} \in W_{kl}$ for $S_k = S$ and $S_l = S'$. $\square$

Let S_k and S_l be two segments of $Y^-(E)$ such that $E \in W_{kl}$. Assume without loss of generality that $S_k < S_l$ in $Y(E_{kl})$. Since E is the first endpoint to be misplaced, we have $S_k \bigcap L(E_{kl}) <_y S_l \bigcap L(E_{kl})$. For convenience, we will say that two segments S_p and S_q have been *exchanged* between E' and E'', for two events E' and E'', if $S_p < S_q$ in $Y^+(E')$ and $S_q < S_p$ in $Y^-(E'')$.

The case where $E \in W_{kl}^{-}(E_{kl})$ cannot cause any difficulty since S_k and S_l cannot be active between E_{kl} and E and therefore S_k and S_l cannot be exchanged between E_{kl} and E, which implies that E is correctly placed with respect to S_k and S_l.

The case where $E \in W_{kl}^{+}(E_{kl})$ is more difficult. E is not correctly placed only if S_k and S_l are not exchanged between E_{kl} and E, i.e., $S_k < S_l$ in both $Y^+(E_{kl})$ and $Y^-(E)$. We shall prove that this is not possible and therefore conclude that S is correctly placed into Y in this case as well.

Assume, for a contradiction, that S_k and S_l have not been exchanged between E_{kl} and E. As E belongs to $W_{kl}^{+}(E_{kl})$, S_k and S_l cannot be adjacent in $Y^-(E)$ since otherwise they would constitute a prime pair and they would have been exchanged. Let $(S_k = S_{k_0}, S_{k_1}, \ldots, S_{k_r}, S_{k_{r+1}} = S_l)$ $(r \geq 1)$ be the subsequence of segments of $Y^-(E)$ occurring between S_k and S_l. Assume that (S_k, S_l) is a pair of intersecting segments such that $E \in W_{kl}^{+}(E_{kl})$, for which r is *minimal* (i.e., for which the above subsequence is shortest). A direct consequence of this definition is the following :

Claim 2 *If S_{k_i} intersects S_k, E cannot belong to $W_{kk_i}^{+}(E_{kk_i})$. Similarly if S_{k_i} intersects S_l, E cannot belong to $W_{k_il}^{+}(E_{k_il})$.*

We distinguish the two following cases:

(i) $E <_y S_{k_i}$. S_{k_i} intersects S_l to the left of $L(E)$ because S_{k_i} and S_l are correctly placed in $Y^+(E_{k_il})$ (E is the first endpoint to be misplaced) and misplaced in $Y^-(E)$. It then follows from Claim 2 that $E \in W_{k_il}^{-}(E_{k_il})$ and therefore $S_l \bigcap L(E_{k_il}) <_y S_{k_i} \bigcap L(E_{k_il})$ since $S_l \bigcap L(E) <_y S_{k_i} \bigcap L(E)$. As E is the first endpoint to be misplaced, we have $S_l < S_{k_i}$ in $Y^+(E_{k_il})$. Moreover, since the pair (S_{k_i}, S_l) is not active (between E_{k_il} and E), and therefore cannot be exchanged, the same inequality holds in $Y^-(E)$, which contradicts the definition of S_{k_i}.

(ii) $S_{k_i} <_y E$. This case is entirely symetric to the previous one. It suffices to exchange the roles of S_k and S_l and to reverse the relations $<$ and $<_y$.

Since a contradiction has been reached in both cases, the lemma is proved. $\square$

We now complete the proof of the theorem. The previous lemma implies that the endpoints are correctly processed. Indeed let E_i be an endpoint. If E_i is a right endpoint, we simply remove the corresponding segment from Y and update the set of active segments. This can be done exactly since predicates of degree ≤ 2 are evaluated correctly. If E_i is a left endpoint, it is correctly placed in Y on the basis of the previous lemma.

The lemma also implies that all pairs that intersect have been processed. Indeed if S_p and S_q are two intersecting segments such that $S_p \bigcap L(A_{pq}) <_y S_q \bigcap L(A_{pq})$ and $S_q \bigcap L(B_{pq}) <_y S_p \bigcap L(B_{pq})$, the lemma shows that $S_p < S_q$ in $Y^+(A_{pq})$ and $S_p > S_q$ in $Y^-(B_{pq})$, which implies that the pair (S_p, S_q) has been processed (Lemma 7).

This concludes the proof of the theorem. $\square$

Remark 3. Handling the degenerate cases does not cause any difficulty and the previous algorithm will work with only minor changes. For the initial sorting of the endpoints, we can take any order relation compatible with the order of their x-coordinates, e.g., the lexicographic order.

Remark 4. Theorem 6 applies directly to pseudo-segments, i.e., curved segments that intersect in at most one point. Lemmas 7 and 8 also extend to the case of monotone arcs that may intersect in more than one point. To be more precise, in Lemma 7, we have to replace "intersect" by "intersect an odd number of times"; Lemma 8 and its proof are unchanged provided that we define $W_{kl}^+(E_{kl})$ (resp., $W_{kl}^-(E_{kl})$) as the subset of W_{kl} consisting of the endpoints E, $E_{kl} <_x E$, such that the slab (E_{kl}, E) contains an odd number (resp. none or an even number) of intersection points. As a consequence, the lazy algorithm (which still uses only Predicates 1, 2 and $2'$) will detect all pairs of arcs that intersect an odd number of times.

Remark 5. For line segments, observe that checking whether a pair of segments is active does not require to know (and therefore to maintain) E_{kl}. In fact, we can replace Condition 3 in the definition of an active pair by the following condition : $S_l <_y F_{kl} <_y S_k$ and $I_{kl} <_x F_{kl}$. If $E_{kl} <_x I_{kl}$, the two definitions are identical and if $I_{kl} <_x E_{kl}$, the pair is not active since, by Lemma 8, Condition 2 of the definition won't be satisfied.

6 Efficient implementations of the lazy algorithm in the predicate arithmetic model

The difficulty to efficiently implement the lazy sweep-line algorithm using only predicates of degree at most 2 (i.e., in the predicate arithmetic model of degree 2) is due to verification of the emptiness condition in Definition 4 and of the condition expressed by Definition 5. One can easily check that various known implementations of the sweep achieve straightforward verification of the emptiness condition by introducing algorithmic complications. The following subsection describes an efficient implementation of the lazy algorithm in the predicate arithmetic model of degree 3. The second subsection improves on this result in a special but important instance of Pb1, namely the case of two sets of non-intersecting segments. The algorithm presented there uses only predicates of degree at most 2.

6.1 Robustness of the standard sweep-line algorithm

We shall run our lazy algorithm under the predicate arithmetic model of degree 3. We then have the capability to correctly compare the abscissae of an intersection and of an endpoint. We refine the lazy algorithm in the following way. Let E_i be the last processed endpoint and let E_{i+1} be the endpoint to be processed next. An active pair (S_k, S_l) that occurs in Y between $Y(E_i)$ and $Y^-(E_{i+1})$ will be processed if and only if its intersection point I_{kl} lies to the right of E_i and not to the right of E_{i+1}. As the slab is free of endpoints in its interior, any pair of adjacent segments encountered in Y (between $Y(E_i)$ and $Y^-(E_{i+1})$) and that intersect within the slab is active. Moreover the intersection points of all prime pairs belong to the slab. It follows that this instance of the lazy algorithm need not explicitly check whether a pair is active or not, and therefore is much more efficient than the lazy algorithm of Section 5. This algorithm is basically what the original algorithm of Bentley-Ottmann * becomes when predicates of degree at most 3 are evaluated (recall that the standard algorithm requires the capability to correctly execute predicates of degree up to 5).

* With the policy concerning multiple detections of intersections that is stipulated at the beginning of Section 5.

We therefore conclude with the following theorem :

Theorem 9 *If Predicates 1, 2 and 3 are evaluated exactly, the standard sweep-line algorithm will solve Pb1 in $O((n+k)\log n)$ time.*

It is now appropriate to briefly comment on the implementation details of the just described modified algorithm. Data structure Y is implemented as usual as a dictionary. Data structure X, however, is even simpler than in the standard algorithm (which uses a priority queue with dictionary access). Here X has a primary component realized as a static dynamic search tree on the abscissae of the endpoints $E_1, \ldots, E_{2n}$. Leaf E_j points to a secondary data structure $\mathcal{L}(E_j)$ realized as a conventional linked list, containing (in an arbitrary order) adjacent intersecting pairs in slab $(E_j, E_{j+1}]$. Remember that each intersecting pair of $\mathcal{L}(E_j)$ is active. Insertion into $\mathcal{L}(E_j)$ is performed at one of its ends and so is access for reporting (when the plane sweep reaches slab $(E_j, E_{j+1}]$). To effect constant-time removal of a pair (S_h, S_k) due to loss of adjacency, however, a pointer could be maintained from a fixed member of the pair (say, the one with smaller left endpoint in lexicographic order) to the record stored in X (notice that the described insertion/removal policy, which guarantees that the elements of X correspond to pairs of adjacent segments in Y, ensures that at most one record is to be pointed to by any member of Y). We finally observe that a segment adjacency arising in Y during the execution of the algorithm must be tested for intersection; however, an intersecting pair of adjacent segments is eligible for insertion into X only as long as the plane sweep has not gone beyond the slab containing the intersection in question. As regards the running time, beside the initial sorting of the endpoints and the creation of the corresponding primary tree in time $O(n \log n)$, it is easily seen that each intersection uses $O(\log n)$ time (amortized), thereby achieving the performance of the standard algorithm.

Finally, we note that if only predicates of degree ≤ 2 are evaluated correctly, the algorithm of Bentley-Ottmann may fail to report the set of intersecting pairs of segments. See Figure 3 for an example.

Remark 6. The fact that the sweep line algorithm does not need to sort intersection points had already been observed by Myers [Mye85] and Schorn [Sch91]. Myers does not use it for solving robustness problems but for developing an algorithm with an expected running time of $O(n \log n + k)$. Schorn uses this fact to decrease

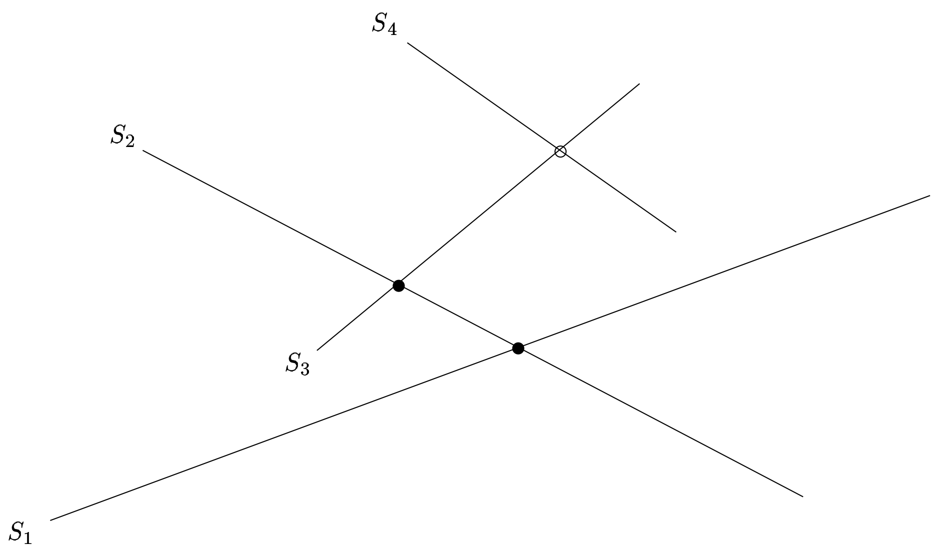

Figure 3: If the computed x-coordinate of the intersection point of S_1 and S_2 is (erroneously) found to be smaller than the x-coordinate of the left endpoint of S_3 and if S_3 and S_4 are (correctly) inserted below S_2 and above S_1 respectively, then the intersection between S_3 and S_4 will not be detected. Observe that the missed intersection point can be arbitrarily far from the intersection point involved in the wrong decision.

the precision required by the sweep line algorithm from five fold to three fold, i.e., Schorn's algorithm uses exact arithmetic of degree 3. Using Theorem 6, we will show in Section 7 that double precision suffices.

6.2 Reporting intersections between two sets of nonintersecting line segments

In this subsection, we consider two sets of line segments in the plane, $\mathcal{S}_b$ (the blue set) and $\mathcal{S}_r$ (the red set), where no two segments in $\mathcal{S}_b$ (similarly, in $\mathcal{S}_r$) intersect. Such a problem arises in many applications, including the union of two polygons and the merge of two planar maps. We denote by n_b and n_r the cardinalities of $\mathcal{S}_b$ and $\mathcal{S}_r$, respectively, and let $n = n_b + n_r$.

Mairson and Stolfi [MS88] have proposed an algorithm that works for arcs of curve as well as for line segments. Its time complexity is $O(n \log n + k)$, which is optimal, and requires $O(n + k)$ space ($O(n)$ in case of line segments). The same asymptotic time-bound has been obtained by Chazelle et al. [CEGS94] and by Chazelle and Edelsbrunner [CE92]. The latter algorithm is not restricted to two sets of nonintersecting line segments. Other algorithms have been proposed by Nievergelt and Preparata [NP82] and by Guibas and Seidel [GS87] in the case where the segments of $\mathcal{S}_b$ (and $\mathcal{S}_r$) are the edges of a subdivision with convex faces. With the exception of the algorithm of Chazelle et al. [CEGS94], all these algorithms construct the resulting arrangement and therefore have degree 4. The algorithm of Chazelle et al. requires to sort the intersection points of two segments with a vertical line passing through an endpoint. Therefore it is of degree 3.

We propose instead an algorithm that computes all the intersections but not the arrangement. This algorithm uses only predicates of degree ≤ 2 and has time complexity $O((n + k) \log n)$.

We say that a point E_i is *vertically visible* from a segment $S_b \in \mathcal{S}_b$ if the vertical line segment joining E_i with S_b does not intersect any other segment in $\mathcal{S}_b$ (the same notion is applicable to $\mathcal{S}_r$). For two intersecting segments $S_b \in \mathcal{S}_b$ and $S_r \in \mathcal{S}_r$, let L be a vertical line to the right of A_{br} such that no other segment intersects L between S_b and S_r (i.e., S_b and S_r are adjacent). We let T_{br} denote the wedge defined by S_b and S_r in the slab between L and $L(I_{kl})$.

Our algorithm is based on the following observation :

Lemma 10 *T_{br} contains blue endpoints if and only if it contains a blue endpoint that is vertically visible from S_b. Similarly, T_{br} contains red endpoints if and only if it contains a red endpoint vertically visible from S_r.*

Proof : The sufficient condition is trivial, so we only prove necessity. Assume without loss of generality that $S_r \bigcap L(A_{br}) <_y S_b \bigcap L(A_{br})$. Let $\mathcal{E}$ be the subset of the blue endpoints that belong to T_{br} and $CH^+(\mathcal{E})$ their upper convex hull. Clearly, all vertices of $CH^+(\mathcal{E})$ are vertically visible from S_b. $\square$

Our algorithm has two phases. The second one is the lazy algorithm of Section 5. The first one can be considered as a preprocessing step that will help to efficiently find active pairs of segments.

More specifically, our objective is to develop a quick test of the emptiness condition based on the previous lemma. The preprocessing phase is aimed at identifying the candidate endpoints for their potential belonging to wedges formed by intersecting adjacent pairs. Referring to $\mathcal{S}_b$ (and analogously for $\mathcal{S}_r$), we first sweep the segments of $\mathcal{S}_b$ and construct for each blue segment S_b the lists $\mathcal{L}_b^-$ and $\mathcal{L}_b^+$ of blue endpoints that are vertically visible from S_b and lie respectively below and above S_b. The sweep takes time $O(n \log n)$ and the constructed lists are sorted by increasing abscissa. Since there is no intersection point, only predicates of degree ≤ 2 are used. The total size of the lists $\mathcal{L}_b^-$, $\mathcal{L}_b^+$, $\mathcal{L}_r^-$, and $\mathcal{L}_r^+$ is $O(n)$.

As mentioned above, the crucial point is to decide whether the wedge T_{br} of a pair of intersecting segments S_b and S_r adjacent in Y contains or not endpoints of other segments. Without any loss of generality we assume that $S_r < S_b$ in Y. If such endpoints exist, then T_{br} contains either a blue vertex of $CH^+(\mathcal{L}_b^- \cap L^+)$ or a red vertex of $CH^-(\mathcal{L}_b^+ \cap L^+)$.

We will show below that, using predicates of degree ≤ 2, the lists can be preprocessed in time $O(n \log n)$ and that deciding whether T_{br} contains or not endpoints can be done in time $O(\log n)$, using only predicates of degree at most 2.

Assuming for the moment that this primitive is available, we can execute the plane sweep algorithm described earlier. Specifically, we sweep $\mathcal{S}_b$ and $\mathcal{S}_r$ simultaneously, using the lazy sweep-line algorithm of Section 5[†]. Each time we detect a pair of (adjacent) intersecting segments S_b and S_r, we can decide in time $O(\log n)$ whether they are "active" or "not active", using only predicates of degree ≤ 2.

We sum up the results of this section in the following theorem :

Theorem 11 *Given n line segments in the plane belonging to two sets $\mathcal{S}_b$ and $\mathcal{S}_r$, where no two segments in $\mathcal{S}_b$ (analogously, in $\mathcal{S}_r$) intersect, there exists an algorithm of optimal degree 2 that reports all intersecting pairs in $O((n+k)\log n)$ time using $O(n)$ storage.*

We now return to the implementation of the primitive described above. Suppose that, for some segment S_i ($i = b$ or r) we have constructed the upper convex hull $\mathrm{CH}^+(\mathcal{L}_i^- \cap L^+)$. Then we can detect in $O(\log n)$ time if an element of a list, say $\mathcal{L}_b^-$ lies above some segment S_r. More specifically, we first identify among the edges of $CH^+(\mathcal{L}_b^- \cap L^+)$ two edges whose slopes are respectively smaller and greater than the slope of S_r. This only requires the evaluation of $O(\log |\mathcal{L}_b^-|)$ predicates of degree 2. It then remains to decide whether the common endpoint E of the two reported edges lies above or below the line containing S_r. This can be answered by evaluating the orientation predicate $\mathrm{orient}(E, A_r, B_r)$.

The crucial requirement of the adopted data structure is the ability to efficiently maintain $\mathrm{CH}^+(\mathcal{L}_i^- \cap L^+)$. To this purpose, we propose the following solution.

The data structure associated with a list $\mathcal{L}_i^-$ ($i = b$ or r) represents the upper convex hull $\mathrm{CH}^+(\mathcal{L}_i^-)$ of $\mathcal{L}_i^-$. (Similarly, the data structure associated to a list $\mathcal{L}_i^+$ represents the lower convex hull $\mathrm{CH}^-(\mathcal{L}_i^+)$ of $\mathcal{L}_i^+$.) This implies that a binary search on the convex hull slopes uniquely identifies the test vertex. Since the elements of each list are already sorted by increasing x-coordinates, the data structures can be constructed in time proportional to their sizes, therefore in $O(n)$ time in total. It can be easily checked that only orientation predicates (of degree 2) are involved in this process. To guarantee the availability of $\mathrm{CH}^+(\mathcal{L}_i^- \cap L^+)$, we have to ensure that our data structure can efficiently handle the deletion of elements. As elements are deleted in order of increasing abscissa, this can be done in amortized $O(\log |S|)$ time per deletion [HS90, HS96]. It follows that preprocessing all lists takes $O(n)$ time, uses $O(n)$ space and only requires the evaluation of predicates of degree ≤ 2.

[†] We can adopt the policy of processing all active pairs before the next endpoint.

7 An efficient implementation of the lazy algorithm under the exact arithmetic model of degree 2

We shall run the lazy algorithm of Section 6.1 under the exact arithmetic model of degree 2, i.e. Predicates 1 and 2 are evaluated exactly but Predicate 3 is implemented with exact arithmetic of degree 2 as explained in Section 4.3. Several intersection points may now be found to have the same abscissa as an endpoint. We refine the lazy algorithm in the following way. Let E_i be the last processed endpoint and let E_{i+1} be the endpoint with an abscissa strictly greater than the abscissa of E_i to be processed next. An active pair (S_k, S_l) will be processed if and only if its intersection point is found to lie to the right of E_i and not to the right of E_{i+1}.

We claim that this policy leads to efficient verification of the emptiness condition. Indeed, the intersections of all prime pairs belong to $(E_i, E_{i+1}]$, because $E_{i+1} \in W^+_{kl}(E_{kl}) \Longrightarrow I_{kl} \leq_x E_{i+1}$, and, by the monotonicity property, I_{kl} will be found to be $\leq_x E_{i+1}$.

The crucial observation that drastically reduces the time complexity is the following. A pair of adjacent segments (S_k, S_l) encountered in Y between $Y(E_i)$ and $Y(E^-_{i+1})$ whose intersection point is found to lie in slab $(E_i, E_{i+1}]$ is active if and only if $E_{i+1} \notin W^-_{kl}(E_{kl})$. Indeed, since I_{kl} is found to be $<_x E_{i+2}$, the monotonicity property implies that $I_{kl} <_x E_{i+2}$. Therefore, when checking if a pair is active, it is sufficient to consider just the next endpoint, not all of them.

Theorem 6 therefore applies. If no two endpoints have the same x-coordinate, the algorithm can use the same data structures as the algorithm in Section 6.1 and its time complexity is clearly the same as for the Bentley-Ottmann's algorithm. Otherwise, we construct X on the distinct abscissae of the endpoints and store all endpoints with identical x-coordinates in a secondary search structure with endpoints sorted by y-coordinates. This secondary structure will allow to determine if a pair is active in logarithmic time by binary search. We conclude with the following theorem :

Theorem 12 *Under the exact arithmetic model of degree 2, the instance of the lazy algorithm described above solves Pb1 in $O((n + k)\log n)$ time.*

8 Conclusion

Further pursuing our investigations in the context of the exact-computation paradigm, in this paper we have illustrated that important problems on segment sets (such as intersection report, arrangement, and trapezoidal map), which are viewed as equivalent under the Real-RAM model of computation, are distinct if their arithmetic degree is taken into account. This sheds new light on robustness issues which are intimately connected with the notion of algorithmic degree and illustrates the richness of this new direction of research.

For example, we have shown that the well-known plane-sweep algorithm of Bentley-Ottmann uses more machinery than strictly necessary, and can be appropriately modified to report segment intersections with arithmetic capabilities very close to optimal and no sacrifice in performance.

Another result of our work is that exact solutions of some problems can be obtained even if approximate (or even random) evaluations of some predicates are performed. More specifically, using less powerful arithmetic than demanded by the application, we have been able to compute the vertices of an arrangement of line segments by constructing an arrangement which may be different from the actual one (and may not even correspond to any set of straight line segments) but still have the same vertex set.

Our work shows that the sweep-line algorithm is more robust than usually believed, proposes practical improvements leading to robust implementations, and provides a better understanding of the sweeping line paradigm. The key to our technique is to relax the horizontal ordering of the sweep. This is one step further after similar attemps aimed though at different purposes [Mye85, MS88, EG89].

A host of interesting open questions remain. One such question is to devise an output-sensitive algorithm for reporting segment intersections with optimal time complexity and with optimal algorithmic degree (that is, 2). It would also be interesting to examine the plane-sweep paradigm in general. For example, with regard to the construction of Voronoi diagrams in the plane, one should elucidate the reasons for the apparent gap between the algorithmic degrees of Fortune's plane-sweep solution and of the (optimal) divide-and-conquer and incremental algorithms.

Acknowledgments

We are indebted to H. Brönnimann for having pointed out an error in a previous version of this paper and to O. Devillers for discussions that lead to Lemma 3. S. Pion, M. Teillaud and M. Yvinec are also gratefully acknowledged for their comments on this work.

References

[ABD+97] F. Avnaim, J.-D. Boissonnat, O. Devillers, F. Preparata, and M. Yvinec. Evaluating signs of determinants using single-precision arithmetic. *Algorithmica*, 17:111–132, 1997.

[Bal95] Ivan J. Balaban. An optimal algorithm for finding segment intersections. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 211–219, 1995.

[BDS+92] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete Comput. Geom.*, 8:51–71, 1992.

[BEPP97] H. Bronnimann, I. Emiris, V. Pan, and S. Pion. Computing exact geometric predicates using modular arithmetic with single precision. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 174–182, 1997.

[BKM+95] Christoph Burnikel, Jochen Könnemann, Kurt Mehlhorn, Stefan Näher, Stefan Schirra, and Christian Uhrig. Exact geometric computation in LEDA. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C18–C19, 1995.

[BMS94] C. Burnikel, K. Mehlhorn, and S. Schirra. On degeneracy in geometric computations. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 16–23, 1994.

[BO79] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28:643–647, 1979.

[Boc07] Maxime Bocher. *Introduction to higher algebra*. Macmillan, 1907.

[Bro81] K. Q. Brown. Comments on "Algorithms for reporting and counting geometric intersections". *IEEE Trans. Comput.*, C-30:147–148, 1981.

[BY97] H. Bronnimann and M. Yvinec. Efficient exact evaluation of signs of determinants. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 166–173, 1997.

[CE92] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39:1–54, 1992.

[CEGS94] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. *Algorithmica*, 11:116–132, 1994.

[Cha91] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6:485–524, 1991.

[Cla92] K. L. Clarkson. Safe and effective determinant evaluation. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 387–395, 1992.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[EG89] H. Edelsbrunner and L. J. Guibas. Topologically sweeping an arrangement. *J. Comput. Syst. Sci.*, 38:165–194, 1989. Corrigendum in 42 (1991), 249–251.

[For85] A. R. Forrest. Computational geometry in practice. In R. A. Earnshaw, editor, *Fundamental Algorithms for Computer Graphics*, volume F17 of *NATO ASI*, pages 707–724. Springer-Verlag, 1985.

[For87] A. R. Forrest. Computational geometry and software engineering: Towards a geometric computing environment. In D. F. Rogers and R. A. Earnshaw, editors, *Techniques for Computer Graphics*, pages 23–37. Springer-Verlag, 1987.

[FV93] S. Fortune and C. J. Van Wyk. Efficient exact arithmetic for computational geometry. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 163–172, 1993.

[Gol91] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 32(1):5–48, March 1991.

[GS87] L. J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Discrete Comput. Geom.*, 2:175–193, 1987.

[HHK89] C. M. Hoffmann, J. E. Hopcroft, and M. T. Karasick. Robust set operations on polyhedral solids. *IEEE Comput. Graph. Appl.*, 9(6):50–59, November 1989.

[Hof89] C. M. Hoffmann. The problems of accuracy and robustness in geometric computation. *IEEE Computer*, 22(3):31–41, March 1989.

[HS90] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. In *Proc. 2nd Scand. Workshop Algorithm Theory*, volume 447 of *Lecture Notes Comput. Sci.*, pages 380–392. Springer-Verlag, 1990.

[HS96] J. Hershberger and S. Suri. Off-line maintenance of planar configurations. *J. Algorithms*, 21:453–475, 1996.

[LPT96] Giuseppe Liotta, Franco P. Preparata, and Roberto Tamassia. Robust proximity queries in implicit Voronoi diagrams. Technical Report CS-96-16, Center for Geometric Computing, Comput. Sci. Dept., Brown Univ., Providence, RI, 1996.

[Mil88] V. J. Milenkovic. Verifiable implementations of geometric algorithms using finite precision arithmetic. *Artif. Intell.*, 37:377–401, 1988.

[Mil89] V. Milenkovic. Double precision geometry: a general technique for calculating line and segment intersections using rounded arithmetic. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 500–505, 1989.

[MS88] H. G. Mairson and J. Stolfi. Reporting and counting intersections between two sets of line segments. In R. A. Earnshaw, editor, *Theoretical*

Foundations of Computer Graphics and CAD, volume 40 of *NATO ASI Series F*, pages 307–325. Springer-Verlag, Berlin, West Germany, 1988.

[Mye85] E. W. Myers. An $O(E \log E + I)$ expected time algorithm for the planar segment intersection problem. *SIAM J. Comput.*, 14:625–637, 1985.

[NP82] J. Nievergelt and F. P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Commun. ACM*, 25:739–747, 1982.

[Pri92] D. Priest. *On properties of floating point arithmetics: numerical stability and the cost of accurate computations.* Ph.D. thesis, Dept. of mathematics, Univ. of California at Berkeley, 1992.

[Sch91] P. Schorn. *Robust Algorithms in a Program Library for Geometric Computation*, volume 32 of *Informatik-Dissertationen ETH Zürich.* Verlag der Fachvereine, Zürich, 1991.

[She96] Jonathan R. Shewchuk. Robust adaptive floating-point geometric predicates. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 141–150, 1996.

[Yap97] C. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 7:3–23, 1997.

Appendix

In connection with reducibility of polynomials over a domain of rationality, we choose the rationals as the latter. "Reducible" means "reducible over the rationals".

We first recall that a general determinant is irreducible if its entries are regarded as independent variables [Boc07]. This suffices to prove directly that the degree of Predicate 2 is 2, and will be crucial for completing the proof of irreducibility of the polynomials pertaining to the other predicates. Use will be made of the following theorem.

Let $p(x_1, \ldots, x_n)$ be a multivariate homogeneous polynomial and let I be a subset of $\{1, \ldots, n\}$, such that for any $j \in I$, x_j is a variable of degree 1 in p. For $i \in \{1, \ldots, n\}$

p can be expressed as

$$p = p_i x_i + p_{i0}$$

where p_i and p_{i0} are polynomials in all variables except x_i. Two polynomials are not considered distinct if they differ just by a multiplicative constant. We have the following :

Theorem 13 *Let $p(x_1, \ldots, x_n)$ be a multivariate homogeneous polynomial with degree at least 3 and $|I| \geq 2$. If for some $i, j \in I$, coefficients p_i and p_j are distinct and irreducible, then p is irreducible.*

Proof. Assume, for a contradiction, that p is reducible. This means that p can be expressed as $p = \phi\psi$, where ϕ and ψ are multivariate homogeneous polynomials over the same variables satisfying $1 \leq degree(\phi), degree(\psi) < degree(p)$. Each degree-1 variable obviously appears in exactly one of the factors. With complete generality, assume that x_i appears in ϕ. We distinguish two cases:

1. $degree(\phi) > 1$. Expanding ϕ around x_i we obtain $\phi = \phi_i x_i + \phi_{i0}$, so that

   $$p = \phi\psi = (\phi_i x_i + \phi_{i0})\psi = \phi_i \psi x_i + \phi_{i0}\psi.$$

   This means that $p_i = \phi_i \psi$, with $degree(\phi_i), degree(\psi) \geq 1$, i.e., p_i is reducible, a contradiction.

2. $degree(\phi) = 1$. Variable x_j appears either in ϕ or in ψ. In the first case, p_i and p_j are both proportional to ψ, i.e., they may differ only by a multiplicative constant and are not distinct, a contradiction.

   In the second case, by an argument analogous to that of Case 1, we reach the conclusion that $p_j = \phi\psi_j$ and that $degree(\psi_j) = degree(p) - degree(\phi) - 1 \geq 3 - 1 - 1 = 1$. Since $p_j = \phi\psi_j$ and $degree(\phi), degree(\psi_j) \geq 1$, p_j is reducible, another contradiction.

This completes the proof of the theorem. □

The calculation of coefficients p_i, p_j corresponds to taking formal derivatives of p with respect to x_i,x_j. Notice that a derivation reduces the degree by 1. The reader may verify, by explicit and not very enlightening calculations, that the following schedules of derivations lead to irreducible 2×2 determinants. The indices conform with the notation of Section 4.1.

Predicate 3: Apply Theorem 13 to the pair (x_1, x_2).

Predicate 4: Apply Theorem 13 first to (x_2, x_3) and then to (y_6, y_7).

Predicate 4′: Apply Theorem 13 first to (x_0, x_1) and then to (x_2, x_3)

Predicate 5: Apply Theorem 13 first to (x_1, x_2), next to (x_7, x_8), and finally to (y_5, y_6).

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399